

Resumen de Organización del Computador 1

escrito en formato de preguntas y respuestas - julio 2022

Hicimos este resumen con el fin de estudiar para el final de la materia.

Nos basamos en los contenidos del libro:

The essentials of computer organization and architecture / Linda Null, Julia Lobur.

- Ramiro Sánchez Posadas y Martino Simón

¿Cuál es la diferencia entre la organización y arquitectura de una computadora?


La organización se refiere a los aspectos físicos de los sistemas informáticos. Consiste en cómo se implementa la arquitectura. Abarca las señales de control, métodos de señalización y tipos de memoria. Responde a la pregunta ¿cómo funciona la máquina?

La arquitectura se refiere a los aspectos lógicos de la implementación del sistema tal como los ve el programador. Se enfoca en la estructura y el comportamiento de la computadora. La arquitectura de la computadora incluye muchos elementos como set de instrucciones y formatos, códigos de operación, tipos de datos, el número y tipos de registros, modos de direccionamiento, métodos de acceso a la memoria principal y varios mecanismos de E/S. La arquitectura afecta directamente a la ejecución lógica de los programas. La arquitectura de la computadora para una máquina dada es la combinación de sus componentes de hardware más su *instruction set architecture* (ISA). Responde a la pregunta ¿cómo se diseña una computadora y qué puede hacer?

¿Qué es la ISA?

La ISA es el set de instrucciones de la arquitectura (instruction set architecture). Es la interfaz que existe entre todo el software que corre en la máquina y el hardware que lo ejecuta. Nos permite “hablarle” a la máquina. Está íntimamente relacionada con la organización del sistema, pues se apoya sobre su implementación en términos de componentes electrónicos. Define la forma en la cual un programador observa la arquitectura del sistema.

La ISA indica qué operaciones pueden realizarse en esta computadora cuando programamos en Assembler o se compila un programa en un lenguaje de alto nivel y se obtiene un programa en Assembler.



Nivel 6	Usuario	Programa ejecutables
Nivel 5	Lenguaje de alto nivel	C++, Java, Python, etc.
Nivel 4	Lenguaje ensamblador	Assembly code
Nivel 3	Software del sistema	Sistema operativo, bibliotecas, etc.
Nivel 2	Lenguaje de máquina	Instruction Set Architecture (ISA)
Nivel 1	Unidad de control	Microcódigo / hardware
Nivel 0	Lógica digital	Circuitos, compuertas, memorias

¿Cuál es la importancia del Principio de Equivalencia Hardware-Software?

Principio de equivalencia de hardware y software: todo lo que se puede hacer con software también se puede hacer con hardware, y todo lo que se puede hacer con hardware también se puede hacer con software.

El Principio de Equivalencia de Hardware y Software nos dice que tenemos una opción. Nuestro conocimiento de la organización y arquitectura informática nos ayudará a tomar la mejor decisión.

Lo que varía es la velocidad en que se lleva a cabo la tarea requerida. Las implementaciones en hardware son casi siempre más rápidas.

Muchas veces un sistema embebido simple tendrá un mejor rendimiento que escribir un programa. Pero para otras ocasiones, será mejor escribir un programa, ya que deberíamos tener una computadora para cada problema que tendríamos que solucionar. En cambio, si escribimos programas con una sola computadora capaz de correrlos, podríamos abarcar todos los problemas.

En cada decisión hay que balancear el costo, rendimiento, consumo energético y velocidad.

Nombrar los tres componentes básicos de una computadora

1. CPU: unidad central de procesamiento para interpretar y ejecutar programas
2. Una memoria principal que almacena datos y programas
3. Un sistema de entrada/salida que envía datos desde y hacia el exterior de la computadora

Ley De Moore

Ley de Moore: La densidad de transistores en un circuito integrado se duplica cada año. La versión actual de la ley dice que la densidad de los chips de silicio se duplica cada 18 meses.

Esta ley no puede mantenerse para siempre, ya que hay limitaciones físicas y financieras que entran en juego. El límite está relacionado generalmente con los costos.

Ley de Rock

El costo de fabricación (no exactamente) de semiconductores se duplica cada cuatro años.

¿Qué tenía la arquitectura de von Neumann que la distinguía de sus predecesores? Nombre las características presentes en una arquitectura de von Neumann. ¿Cómo funciona el ciclo fetch-decode-execute?

El modelo de Von Neumann

Antes programar era tanto una hazaña de ingeniería eléctrica como un ejercicio de diseño de algoritmos. Se utilizaban cables para interconectar los componentes de la máquina y cada vez que se quería ejecutar un programa nuevo se debía modificar el cableado de la máquina.

Fue John Von Neumann quien publicó y popularizó la idea de que una computadora debía tener una memoria en la cual se almacenaran los programas y de esa forma no se tuvieran que reconectar cables cada vez que apareciese un programa nuevo de uno u otro investigador. Desde un punto de vista teórico, esta idea responde a la máquina universal de Turing y data de 1936, publicada por Alan Turing.

Todas las computadoras con programas almacenados se conocen como sistema de Von Neumann que utilizan la arquitectura de Von Neumann.

El modelo de Von Neumann se caracteriza por lo siguiente:

1. Los programas y datos se almacenan en la misma memoria sobre la que se puede leer y escribir. Los datos de la memoria son datos que efectivamente se van a usar para operar o instrucciones que uno va a ejecutar con datos que pueden estar en la misma memoria o en otro lugar; ciertos datos pueden representar un programa si es que la computadora decidiera que en esa posición y esos bits representan una instrucción. Si el PC apunta a una dirección en memoria o apuntó alguna vez a esa dirección entonces es una instrucción, porque alguna vez fue cargada, decodificada y ejecutada.
2. La operación que va a realizar la computadora depende del estado actual de la memoria, es decir, lo que está guardado en cada celda y a qué celda se está apuntando.
3. El contenido de la memoria es accedido a través de una posición.

4. La ejecución de los programas de la computadora es secuencial, a menos que en estos se indique lo contrario: se va accediendo a cada celda de memoria en orden, respecto a las posiciones y se ejecuta lo que corresponda. Puede ser que dentro de las ejecuciones suceda que se salteen posiciones de memoria debido a las operaciones que se tienen que realizar. Es por eso que la ejecución de algunos programas puede no ser secuencial.

Arquitectura de Von Newman

Basándose en el modelo de Von Neumann, la arquitectura estándar está formada por tres componentes principales (sistemas de hardware):

1. La CPU: (unidad central de procesamiento) está formada por la unidad de control (UC), ALU (unidad aritmética lógica), Registros (pequeños espacios de almacenamiento) y un PC (program counter). Está relacionada a todo lo que tiene que ver con la lógica de operar datos, almacenarlos internamente para poder operarlos (registros) y orquestar todos los componentes que se tengan.
2. Una memoria: donde se almacenan los programas y datos. Los datos en memoria se almacenan en sistema binario.
3. Un sistema de entrada/salida: la computadora está pensada para poder interactuar con datos que entran y salen, por lo tanto, tiene que existir este subsistema de I/O. Este se encarga de la interconexión externa a la computadora. Parte de lo que llega como datos a la memoria es resuelto por este subsistema.

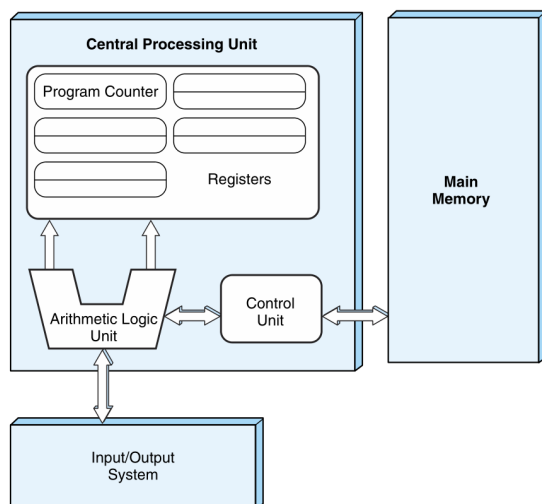


FIGURE 1.4 The von Neumann Architecture

La arquitectura tiene la particularidad de que existe un camino único (ya sea físico o lógico) desde donde se conecta la unidad de control con la memoria. Esta unicidad del camino fuerza la alternación entre ciclos de instrucción y ejecución que se denomina *cuello de botella de Von Neumann*.

La CPU opera instrucciones indicadas en la memoria a través del ciclo de instrucción. La CPU viene con un reloj incorporado que sirve para la realización de operaciones cíclicas desde que se prende hasta que se apaga la computadora. Esta arquitectura ejecuta programas en lo que se conoce como el ciclo de ejecución de Von Neumann, el cual también se lo llama el ciclo de fetch-decode-execute. Este se divide en tres partes:

1. Fetch: se obtiene la instrucción almacenada en la dirección de memoria que indica el PC y se incrementa el PC, con la ALU o con un incrementador propio; se traen las instrucciones para que la CPU pueda ejecutarlas.
2. Decode: se decodifica la instrucción traída de memoria en un lenguaje que la ALU pueda entender; en caso de que sea necesario, se buscan y obtienen los operandos faltantes desde la memoria y se los coloca en los registros correspondientes, para poder realizar la ejecución.
3. Execute: se realiza cuando ya se tiene la instrucción y todo lo necesario para poder ejecutarla almacenado en algún lugar; la ALU realiza la operación que corresponda, si es necesario, y se coloca en ese caso el resultado en donde sea indicado por la instrucción (algún registro o memoria)

Las ideas presentes en la arquitectura de von Neumann se han ampliado para que los programas y los datos almacenados en un medio de almacenamiento de acceso lento, como un disco duro, se puedan copiar a un medio de almacenamiento volátil de acceso rápido, como la RAM, antes de su ejecución. Esta arquitectura también se ha simplificado en lo que actualmente se denomina *system bus model*. El bus de datos mueve datos de la memoria principal a los registros de la CPU (y viceversa). El bus de direcciones (address) contiene la dirección de los datos a los que está accediendo actualmente el bus de datos. El bus de control transporta las señales de control necesarias que especifican cómo se llevará a cabo la transferencia de información.

Otras mejoras a la arquitectura de von Neumann incluyen el uso de registros de índice para el direccionamiento, la adición de datos de punto flotante, el uso de interrupciones y E/S asíncronas, la adición de memoria virtual y la adición de registros generales.

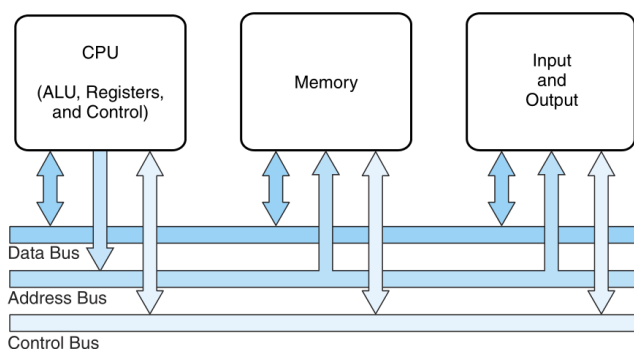


FIGURE 1.5 The Modified von Neumann Architecture, Adding a System Bus

Nombrar y Explicar los 7 Niveles Jerárquicos de Abstracción de las Computadoras Modernas

Es muy difícil hacer un salto directo queriendo relacionar lo que ocurre en los circuitos de la computadora con lo que uno escribe en lenguaje de alto nivel. Para resolver este problema nos sirve pensar en distintos niveles de abstracción, en donde cada uno cumple una función específica y se apoya sobre una computadora hipotética que asumimos que anda y llamamos máquina virtual. Cada máquina virtual ejecuta un set de instrucciones particular, recurriendo a las máquinas de niveles inferiores cuando sea necesario. Es decir, usamos los lenguajes de cada nivel asumiendo que están creados en función de otros más atómicos y que eso funciona.

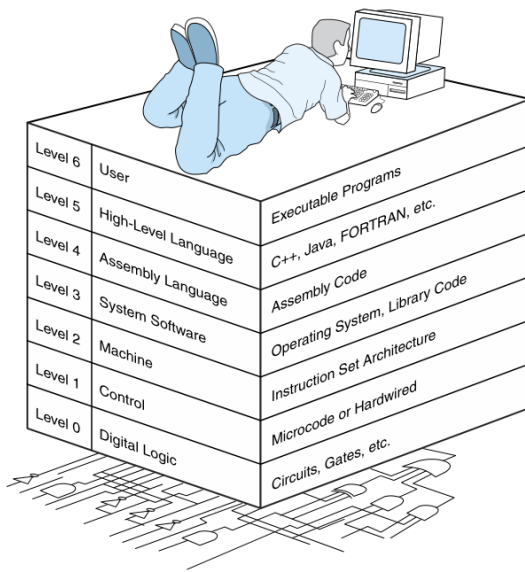


FIGURE 1.3 The Abstract Levels of Modern Computing Systems

El nivel 6 es el del Usuario. Es el nivel al cual estamos más acostumbrados a utilizar. Se compone de aplicaciones como: editores de texto, paquetes gráficos, videojuegos, etc. Los niveles inferiores son prácticamente invisibles desde este nivel.

El nivel 5, Lenguajes de alto nivel, consiste de lenguajes como C, C++, FORTRAN, Lisp, etc. Estos lenguajes deben ser traducidos (usando un compilador o intérprete) a un lenguaje que la computadora pueda entender. Lenguajes compilados se traducen a lenguaje de ensamblador para luego ser ensamblados a código de máquina. Desde este nivel también se ve muy poco de los niveles inferiores.

El nivel 4, lenguaje de ensamblador, recibe el lenguaje de alto nivel compilado, el cual se traduce a lenguaje de ensamblador y luego se realiza una traducción uno a uno a lenguaje de máquina (una instrucción de lenguaje de ensamblador se traduce exactamente a una instrucción de lenguaje de máquina).

El nivel 3, el del sistema de software, se encarga de las instrucciones del sistema operativo. Es responsable de la multiprogramación, proteger la memoria, sincronizar procesos y otras funciones importantes. Es usual que las instrucciones traducidas de lenguaje de ensamblador a lenguaje de máquina pasen por este nivel sin modificaciones.

El nivel 2, el nivel de máquina o Instruction Set Architecture (ISA) consiste del lenguaje de máquina reconocido por la arquitectura particular de la computadora.

El nivel 1, el del Control, es donde la Unidad de Control se asegura que las instrucciones se decodifiquen y ejecuten correctamente y que los datos se muevan en determinado momento, al lugar que correspondan. La Unidad de Control interpreta las instrucciones escritas en lenguaje de máquina que fueron transmitidas por los niveles superiores.

Las unidades de control pueden ser diseñadas de dos maneras: cableada (hardwired) o microprogramadas. En las cableadas, las señales de control son emitidas por bloques de componentes de lógica digital. Estas señales dirigen todos los datos y tráfico de instrucciones a las partes del sistema correspondientes. Este diseño tiene la ventaja de que es muy veloz porque son componentes físicos, pero tiene la desventaja de que es dificultoso modificarlo.

La otra opción para diseñar la Unidad de control es utilizar microprogramas. Estos son escritos en un lenguaje de bajo nivel que se implementa directamente en hardware. Al microprograma se le suministra las instrucciones de máquina del nivel 2. Luego, se activan los componentes de hardware correspondientes para ejecutar la instrucción original. Una instrucción de lenguaje de máquina se suele traducir en varias instrucciones de microcódigo. Los microprogramas son populares porque pueden modificarse fácilmente, pero tiene la contra de que son más lentos, ya que deben pasar por una traducción previa.

El nivel 0, Lógica Digital, es donde encontramos los componentes físicos de la computadora: compuertas y cables. Son los bloques fundamentales de todas las computadoras. Son las implementaciones de la lógica matemática.

¿Qué se entiende por computación paralela?

Lo que realiza la computación paralela es usar la idea de más de una computadora en lugar de intentar desarrollar una computadora más rápida y poderosa que a veces es más complicado. Consiste en utilizar varios procesadores en lugar de utilizar uno solo mejor y más rápido. De todas formas, la computación paralela tiene sus límites, ya que al aumentar la cantidad de procesadores, también aumentan los costos de administrar como se distribuyen las tareas a esos procesadores. Algunos sistemas de procesamiento en paralelo requieren procesadores adicionales solo para administrar el resto de los procesadores y los recursos que se les asignan. No importa cuántos procesadores se coloquen en un sistema, o cuántos recursos se les asignen, de alguna manera, en algún lugar, se desarrollará un cuello de botella. Lo mejor que podemos hacer para remediar esto es asegurarnos de que las partes más lentas del

sistema sean las que se usen menos.

Ley de Amdahl

Esta ley establece que la mejora del rendimiento posible para una mejora dada está limitada por el uso que se le da a la característica mejorada. Es decir, la aceleración de la computadora dada por una mejora de un componente, depende tanto de la aceleración del componente mismo como por la frecuencia en que este es utilizado. La premisa subyacente es que cada algoritmo tiene una parte secuencial que, en última instancia, limita la aceleración que se puede lograr mediante la implementación de multiprocesadores.

$$S = \frac{1}{(1 - f) + f/k}$$

where

S is the speedup;

f is the fraction of work performed by the faster component; and

k is the speedup of a new component.

Donde S es la aceleración,

f es la fracción de trabajo realizada por el componente

K es la aceleración de un nuevo componente

¿Qué es el overflow y cómo se detecta?

El overflow ocurre en la representación binaria cuando el resultado de una operación aritmética está fuera del rango de precisión permisible para el número dado de bits. En general, se evidencia al obtener un resultado erróneo luego de realizar una operación.

¿Por qué es importante la comprensión del álgebra booleana para los informáticos?

El álgebra booleana es un álgebra para la manipulación de objetos que puede tomar solo dos valores, generalmente verdadero y falso, aunque puede ser cualquier par de valores. Debido a que las computadoras se construyen como colecciones de interruptores que

están "encendidos" o "apagados", el álgebra booleana es una forma muy natural de representar información digital. En realidad, los circuitos digitales usan voltajes bajos y altos, pero nosotros utilizamos 0 y 1. Es común interpretar el valor digital 0 como falso y el valor digital 1 como verdadero aunque esto no necesariamente debe ser así.

La relación entre la lógica booleana y los componentes físicos reales de cualquier sistema informático es muy estrecha. Comprender cómo la lógica booleana afecta el diseño de varios componentes del sistema informático permite utilizar, desde una perspectiva de programación, cualquier sistema informático con mayor eficacia.

Las expresiones booleanas se representan con diagramas lógicos. Estos representan la implementación física de la expresión, o el circuito digital. Por lo tanto, utilizamos el álgebra de Bool para analizar y diseñar estos circuitos digitales.

¿Por qué es importante que las expresiones booleanas sean minimizadas en el diseño de los circuitos digitales?

Existe una correspondencia entre una expresión booleana y su implementación mediante circuitos eléctricos. Por lo tanto, los términos innecesarios en la expresión conducen a componentes innecesarios en el circuito físico, que a su vez produce un circuito subóptimo y que ocupa mayor espacio.

¿Cuál es la diferencia entre circuitos secuenciales y combinacionales?

Los circuitos combinacionales son aquellos en los que una salida siempre se basa completamente en las entradas dadas. Es decir, la salida de estos circuitos es una función de las entradas, y la salida es únicamente determinada por los valores de las entradas en un momento dado. La principal debilidad de estos circuitos es que no existe el concepto de almacenamiento, no tienen memoria.

Es por esta razón que se crearon los circuitos secuenciales. Estos definen su salida en función de sus entradas actuales y de entradas anteriores. Para recordar entradas anteriores, los circuitos secuenciales deben tener algún tipo de elemento de almacenamiento. Por lo general, este elemento de almacenamiento se denomina flip-flop. El estado de este flip-flop depende de las entradas anteriores. Por lo tanto, la salida pendiente depende tanto de las entradas actuales como del estado actual del circuito. Para "recordar" un estado pasado, los circuitos secuenciales se basan en un concepto llamado retroalimentación. Esto simplemente significa que la salida de un circuito se retroalimenta como una entrada al mismo circuito.

¿Qué flip-flop da una representación real de la memoria física de la computadora?

El flip-flop al que se refiere la pregunta es el flip-flop-D (data). Este circuito es una modificación del flip-flop-RS, ya que consiste en una señal que alimenta tanto la entrada R como la entrada S. La señal es “normal” en la entrada S y negada para la entrada R. Este flip-flop permite almacenar un bit de información y es por eso que representa la memoria física de la computadora.

¿Cuál es la función de la CPU?

La CPU es la responsable de traer las instrucciones de los programas que están en la memoria, decodificar cada instrucción y ejecutar la correspondiente secuencia de operaciones de forma correcta.

Todas las computadoras tienen una unidad de procesamiento central (CPU). Esta unidad se divide en dos partes: el **datapath** y la **unidad de control**.

El **datapath** es una red compuesta por unidades de almacenamiento (registros) y unidades aritméticas y lógicas (para realizar operaciones con datos) conectados a buses (capaces de mover datos de un lugar a otro), todo sincronizado por un clock.

La **unidad de control** es un módulo responsable de ordenar la secuencia de operaciones, asegurándose de que los datos correctos estén en el momento indicado, en el lugar indicado.

¿Qué hace la unidad de control?

La unidad de control es como el “gestor de tráfico en la CPU”. Supervisa la ejecución de todas las instrucciones y la transferencia de toda la información. La unidad de control extrae instrucciones de la memoria, las decodifica, asegurándose de que los datos estén en el lugar correcto en el momento correcto, indica a la ALU qué registros usar, qué servicios interrumpir, y enciende los circuitos correctos de la ALU para realizar la operación deseada. Además, la unidad de control utiliza el PC para encontrar en memoria la siguiente instrucción a ejecutar, y controla los registros de estado (flags) para realizar un seguimiento de los overflow, carry, borrow y similares.

¿Qué son los registros, dónde se ubican y qué tipos de registros existen?

Un registro es un dispositivo de hardware que almacena datos binarios. Están ubicados en el procesador, por lo que se puede acceder a la información muy rápidamente. Los registros se utilizan en los sistemas informáticos como lugares para almacenar una amplia variedad de datos, como direcciones, PC o datos necesarios para la ejecución de programas.

Algunos registros se especifican como de "propósito especial" y pueden contener solo datos, solo direcciones o solo información de control. Otros son más genéricos y pueden contener datos, direcciones e información de control en varios momentos.

En los sistemas informáticos modernos, existen muchos tipos de registros especializados: registros para almacenar información, para cambiar valores, para comparar valores, registros que cuentan, que almacenan valores temporales, registros de índice para controlar el ciclo del programa, registros de puntero de pila para administrar pilas de información para procesos, registros de estado para mantener el estado o el modo de operación (como condiciones de overflow, carry o cero) y registros de propósito general que son los registros disponibles para el programador. La mayoría de las computadoras tienen conjuntos de registros y cada conjunto se usa de una manera específica.

¿Cómo sabe la ALU qué función debe realizar?

La ALU sabe qué operación tiene que realizar porque es controlada por señales desde la Unidad de Control. La unidad de aritmética lógica lleva a cabo operaciones lógicas (como comparaciones) y operaciones aritméticas (como sumas o multiplicaciones) necesarias durante la ejecución del programa. Las operaciones realizadas en la ALU suelen afectar en los bits del registro de estado (overflow, carry, etc.).

¿Por qué usualmente el bus es un cuello de botella de comunicaciones?

El bus es un conjunto de cables que funcionan de manera conjunta como un camino para la transmisión de datos entre múltiples subsistemas dentro del sistema. Son de bajo costo y versátiles. En cualquier momento determinado, solo un dispositivo (ya sea registro, ALU, etc) puede utilizar el bus. Por lo tanto, hay un cuello de botella en la comunicación, ya que hasta que ese dispositivo no haya dejado de utilizarlo, ningún otro podrá. También la velocidad del bus se ve afectada tanto por su longitud como por el número de dispositivos que lo comparten. Normalmente, los dispositivos se dividen en dos categorías: *master* y *slave*. Donde el master es quien inicia la acción y el slave es quien responde al pedido del master.

¿Por qué es importante tener un protocolo de bus?

En general, un mismo bus puede ser utilizado por varios dispositivos simultáneamente. Es por eso que es necesario tener un protocolo de uso del bus. Es decir, un conjunto de reglas que permita determinar en cada momento qué dispositivo se puede conectar al bus y realizar la transferencia que sea necesaria. De otra forma, si dos dispositivos comparten el mismo bus, se podría producir un cortocircuito, por ejemplo.

Explique las diferencias entre data buses, address buses y control buses.

Las líneas de un bus dedicadas a mover datos se llaman data bus. Estas contienen la información que debe ser movida de un lugar a otro.

Las líneas de control indican qué dispositivo tiene permiso de hacer uso del bus y para qué propósito (leer o escribir desde memoria o el uso por parte de algún dispositivo de entrada/salida, por ejemplo). Además, estas líneas transfieren admisiones de pedido de uso del bus, interrupciones y señales de sincronización del clock.

Las líneas de dirección (address lines) indican el lugar (en memoria, por ejemplo) desde donde se tienen que escribir o leer los datos.

Las power lines proveen la energía eléctrica necesaria.

Cada tipo de transferencia ocurre en un *ciclo de bus*, el tiempo entre dos ticks del reloj del bus.

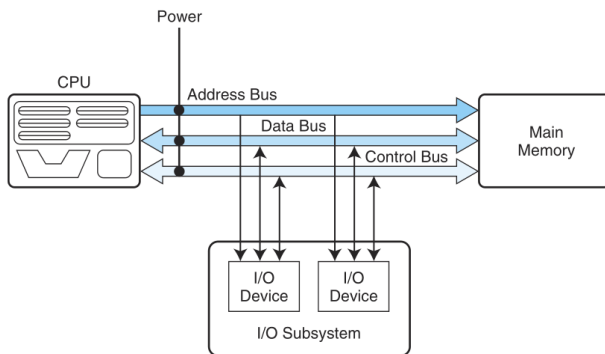


FIGURE 4.2 The Components of a Typical Bus

¿Qué diferencias hay entre el bus síncrono y asíncrono?

Los buses síncronos tienen un reloj y los eventos suceden solo en los ticks de ese reloj. Es decir, la secuencia de eventos es controlada por el reloj. Todos los dispositivos están sincronizados por el ritmo del reloj. El tiempo de un ciclo de bus no debe ser más corto que el tiempo que lleva pasar la información por el bus. Por ejemplo, los buses que recuperan datos de memoria son síncronos, ya que no ocurren situaciones del tipo “la memoria está fuera de línea” o errores que ocurren con dispositivos periféricos.

Mientras que en los buses asíncronos, las líneas de control coordinan las operaciones y un complejo sistema llamado *handshaking protocol* debe ser utilizado para aplicar la sincronización. De todas formas, sigue habiendo un reloj que sincroniza las señales que controlan la transferencia de información. Por ejemplo, los buses de E/S son asíncronos, ya que los dispositivos no siempre están listos para procesar una transferencia.

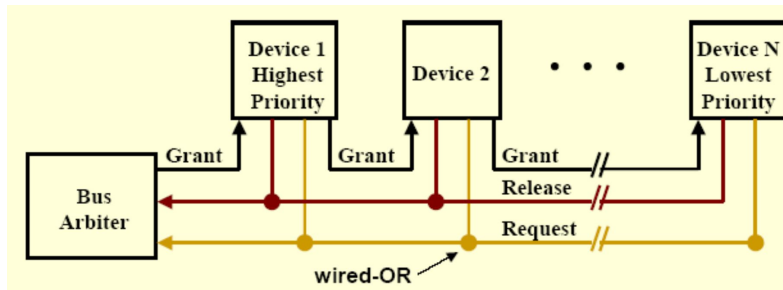
La diferencia principal entre uno y otro es que se sepa contar o no cuántos pulsos de reloj hay que esperar para cada uno de los eventos.

¿Cuáles son los cuatro tipos de arbitraje del bus?

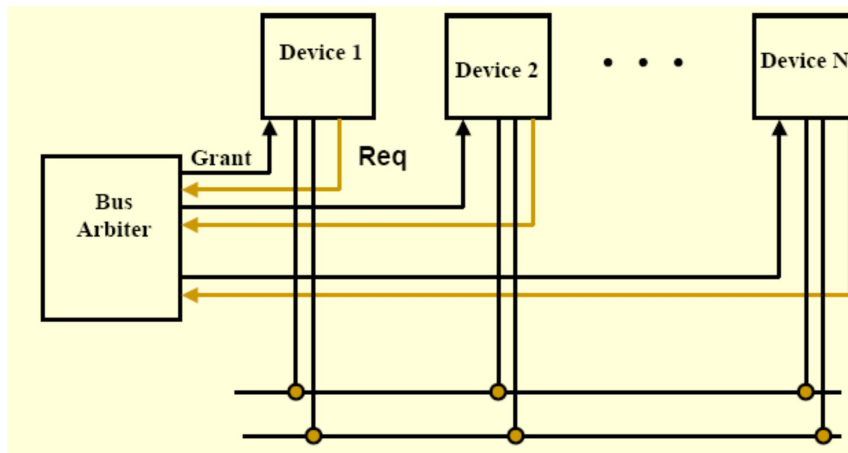
En sistemas con más de un dispositivo master se necesita un arbitraje del bus. De esa manera se le da prioridad a ciertos dispositivos master mientras que se asegura que se le siga dando importancia a los dispositivos de menor prioridad.

Los esquemas de arbitraje del bus caen en cuatro categorías:

1. **Daisy chain arbitration:** se utiliza una línea de control conectada desde el dispositivo con mayor prioridad hasta el de menor. En esa línea de control se utiliza un sistema de pedido de uso del bus. No es justo porque los últimos dispositivos difícilmente puedan acceder al uso del bus.



2. **Centralized parallel arbitration:** cada dispositivo tiene una línea de control de pedido al bus y un árbitro centralizado selecciona quién puede utilizar el bus. Puede haber cuellos de botella.



3. **Distributed arbitration using self-selection:** los dispositivos mismos determinan quién tiene la prioridad mayor y quién debe usar el bus.

4. **Distributed arbitration using collision detection:** cada dispositivo tiene el permiso de hacer un pedido al bus. Si el bus detecta alguna colisión (múltiples pedidos en simultáneo), el dispositivo debe hacer otro pedido.

¿Cuál es la función de una interfaz de entrada-salida (I/O)?

Los dispositivos de entrada y salida nos permiten comunicarnos con la computadora.

Entrada-salida (I/O) es la transferencia de datos entre la memoria principal y varios periféricos de I/O. Los dispositivos de entrada, como teclado y mouse, nos permiten ingresar datos.

Los de salida, como monitor e impresora, nos permiten obtener información de la computadora.

La interfaz de entrada-salida se encarga de esta transmisión de datos, ya que los dispositivos no están conectados directamente a la CPU. La interfaz convierte las señales del sistema de bus a un formato entendible para el dispositivo al cual tiene que llegar. La CPU se comunica con estos dispositivos externos a través de los registros de entrada/salida.

¿Qué hace el ensamblador?

El trabajo del ensamblador es convertir el lenguaje de ensamblador en lenguaje de máquina (que consiste en su totalidad de valores binarios, tiras de ceros y unos). Los ensambladores toman un programa en lenguaje de ensamblador, el cual es una representación simbólica de los números binarios, y lo convierte en instrucciones binarios, o el equivalente en código de máquina. El ensamblador lee un archivo fuente (source file, programa assembly) y produce un archivo objeto (object file, código de máquina).

¿Qué es un sistema embebido? ¿En qué se diferencia de una computadora normal?

Los sistemas embebidos son sistemas en los cuales la computadora se integra a un dispositivo que típicamente no es una computadora. Deben ser reactivos y usualmente se encuentran en ambientes con tiempos limitados. Se diseñan para realizar una instrucción o un set específico de instrucciones. Ejemplos de estos sistemas son: Electrónica (celulares- no smartphones-, cámaras), electrodomésticos (lavarropas, microondas, lavadoras), automóviles, instrumentos médicos (monitores del corazón).

Los compiladores suelen ocultar información acerca del costo (en tiempo) de varias operaciones y a los programadores se les dificulta saber exactamente como sus programas se ejecutarán. El lenguaje de ensamblador le permite al programador acercarse a la arquitectura y así tener un mayor control para cumplir ciertas operaciones, dentro de ciertos parámetros

específicos en un espacio limitado. Es por eso que se utiliza programar el software para sistemas embebidos directamente en lenguaje de ensamblador.

La diferencia principal con una computadora se basa en el principio de equivalencia entre hardware y software. Un sistema embebido es un sistema que suele realizar tareas específicas y que, por lo tanto, tienen un set de instrucciones particular que se puede modelar con hardware. Por otro lado, la computadora es un sistema que puede contener varios programas y puede realizar más funciones al no tener un propósito específico. Tiene un set de instrucciones más amplio y está ligada al uso del software para su funcionamiento.

¿Qué es una pila? ¿Por qué es importante para la programación?

La pila es una estructura de datos que mantiene una lista de elementos a los que se puede acceder desde un solo extremo. La importancia de tener una pila está relacionada a que es una estructura de fácil acceso.

Compare máquinas CISC con máquinas RISC

Las siglas CISC corresponden a Complex Instruction Set Computer. Es decir, una computadora con un conjunto de instrucciones complejo (por ejemplo, la familia x86 de arquitecturas Intel). Las siglas RISC corresponden a Reduced Instruction Set Computer. Es decir, una computadora con un conjunto de instrucciones reducido (por ejemplo, la familia Pentium y las arquitecturas MIPS).

Las máquinas CISC tienen una gran cantidad de instrucciones, de longitud variable, con diseños complejos. Muchas de estas instrucciones son bastante complicadas y realizan múltiples operaciones cuando se ejecuta una sola instrucción (por ejemplo, es posible hacer bucles usando una sola instrucción en lenguaje ensamblador). Además, las arquitecturas CISC incluyen una gran cantidad de instrucciones que acceden directamente a la memoria. Debido a la variedad de tipo de instrucciones, la cantidad de ciclos de reloj por instrucción es variable. Esto hace que la decodificación de las mismas sea más lenta.

El problema básico con las máquinas CISC es que un pequeño subconjunto de instrucciones CISC complejas ralentiza considerablemente los sistemas. Los diseñadores decidieron volver a una arquitectura menos complicada y cablear un conjunto de instrucciones pequeño (pero completo) que se ejecutaría extremadamente rápido. Esto significaba que sería responsabilidad del compilador producir un código eficiente para la ISA. De esta idea nacen las máquinas de tipo RISC.

RISC es un nombre inapropiado. Es cierto que el número de instrucciones se reduce. Sin embargo, el principal objetivo de las máquinas RISC es simplificar las instrucciones para que puedan ejecutarse más rápidamente. La característica principal es que cada instrucción realiza solo una operación, todas tienen el mismo tamaño, hay poca diversidad de tipos, lo que agiliza el proceso de decode y genera que todas las instrucciones tomen la misma cantidad de ciclos de reloj. Además, todas las operaciones aritméticas deben realizarse entre registros (los datos en la memoria no se pueden usar como operandos). Únicamente las instrucciones de *load and store* pueden acceder a los contenidos de memoria. Prácticamente, todos los nuevos conjuntos

de instrucciones (para cualquier arquitectura) desde 1982 han sido RISC, o algún tipo de combinación de CISC y RISC.

Una diferencia importante entre CISC y RISC es la dificultad de implementación del pipeline. En RISC es mucho más fácil debido a que la longitud de las instrucciones suele ser fija y el control microprogramado se suele reemplazar por un control cableado.

Nombrar y explicar los diseños que existen para los sets de instrucciones

Las ISAs se miden por varios factores: la cantidad de espacio que requiere un programa; la complejidad del conjunto de instrucciones, en términos de la cantidad de decodificación necesaria para ejecutar una instrucción y la complejidad de las tareas realizadas por las instrucciones; la extensión de las instrucciones; y el número total de instrucciones.

Las cosas a considerar al diseñar una ISA consisten en:

1. Las instrucciones breves suelen ser mejores porque ocupan menos espacio en memoria. Sin embargo, esto limita el número de instrucciones porque está acortado el tamaño y número de operandos
2. Las instrucciones de longitud fija son más fáciles de decodificar pero desperdician espacio
3. La organización de la memoria afecta al formato de las instrucciones. El direccionamiento y el tamaño de palabra puede dificultar el acceso por bytes
4. Una instrucción de longitud fija no implica necesariamente un número fijo de operandos
5. Tipos diferentes que existen de modos de direccionamiento
6. Big Endian vs. Little Endian
7. Cantidad de registros, cómo se organizan y cómo se almacenan los operandos en la CPU

¿Por qué importa el “endian-ness”? Ejemplo con valor hexadecimal 98765432.

El término *endian* hace referencia al tipo de ordenamiento de bytes en la arquitectura de una computadora. Prácticamente, todas las computadoras modernas tienen direccionamiento a byte y deben tener un estándar para almacenar información que requiera más de un byte.

Un estándar es Little Endian: los bytes en una posición menor son menos significativos.

El otro es Big Endian: los bytes más significativos están en las direcciones menores.

El valor hexadecimal 98765432 sería...

Little: 32-54-76-98

Big: 98-76-54-32

El big endian se lee de una manera más natural para las personas. Además, se puede saber si el número es positivo o negativo fácilmente.

Las operaciones aritméticas de alta precisión son más rápidas y fáciles en little endian.

Little endian permiten leer y escribir palabras en direcciones impares, mientras que todas las direcciones de las arquitecturas con big endian tienen que cumplir los límites que establezca el

tamaño de la palabra (es decir, si una palabra es de 2 o 4 bytes, entonces siempre debe empezar en direcciones de números pares), lo cual genera una pérdida de espacio. Cualquier programa que escriba o lea datos de un archivo, debe tener en cuenta que ordenamiento de bytes utiliza esa máquina.

Diferencia y preferencia en ciertas situaciones de una arquitectura diseñada con: Pila (Stack), Acumulador, registros de propósito general.

Hay tres opciones a la hora de elegir un diseño con el cual la CPU almacenará los datos.

Las arquitecturas diseñadas con una **pila** utilizan una pila para ejecutar las instrucciones, y los operandos se encuentran (implícitamente) en la parte superior de la pila. Aunque las máquinas con este diseño tienen una buena densidad de código y un modelo simple de evaluación de expresiones, no se puede acceder aleatoriamente a la pila, lo cual dificulta generar código eficiente.

Las arquitecturas de **acumulador**, utilizan un operando implícito en el acumulador, lo cual minimiza la complejidad interna de la máquina y permite instrucciones cortas. Pero debido a que el acumulador es solo almacenamiento temporal, el tráfico de la memoria es muy alto.

Los modelos que plantean una arquitectura de **registros de propósito general** son los de mayor aceptación en la actualidad. Estos conjuntos de registros son más rápidos que la memoria, son fáciles de utilizar por los compiladores y pueden usarse eficientemente. Adicionalmente, los costos de hardware han disminuido mucho, permitiendo agregar una gran cantidad de registros con un costo mínimo.

Si el acceso a memoria es rápido, un diseño basado en pila puede funcionar bien; si la memoria es lenta, es mejor utilizar registros. Utilizar registros hace que todos los operandos tengan que ser nombrados (es decir, escribirlos explícitamente), lo cual genera instrucciones más largas, causando tiempos más prolongados de fetch y decode.

¿Cómo se diferencian las arquitecturas de memoria-memoria, registro-memoria y load-store?

Las arquitecturas de propósito general pueden dividirse en tres clasificaciones dependiendo de donde se encuentran sus operandos. Las arquitecturas de **Memoria-memoria** pueden tener dos o tres operandos en memoria, sin necesitar que haya un operando que esté en un registro.

Las arquitecturas **registro-memoria** requieren de una mezcla, donde al menos un operando está en un registro y otro en memoria.

Las arquitecturas **load-store** requieren que los datos se muevan a registro antes de realizar algún tipo de operación sobre ellos.

¿Qué es un modo de direccionamiento? Explique todos los modos.

Los modos de direccionamiento nos permiten especificar donde se encuentran los operandos de las instrucciones. Un modo de direccionamiento puede especificar una constante, un registro o una posición en memoria. Ciertos modos permiten direcciones más cortas y algunos permiten determinar dinámicamente el lugar del operando actual, también llamado la *dirección efectiva* del operando. Los modos de direccionamiento básicos son:

Direccionamiento inmediato se llama de esa forma porque el valor al que se hace referencia es lo que sigue al opcode en la instrucción. Es decir, los datos con los que se operan son parte de la instrucción; esto lo hace de un modo veloz.

Direccionamiento directo se llama de esta forma porque el valor al cual se hace referencia se obtiene especificando su dirección en memoria directamente en la instrucción. Suele ser veloz, ya que el valor se puede acceder rápidamente. Es flexible porque el valor a cargar desde memoria puede ser cualquier dato.

En **Direccionamiento de Registro** se utiliza un registro para especificar el operando. El campo de dirección contiene una referencia a un registro. Los contenidos dentro del registro se son utilizados como operandos.

En **Direccionamiento indirecto** se especifica en el campo de dirección de la instrucción, una dirección de memoria que se usa como puntero. La dirección efectiva del operando es el valor de esta dirección de memoria. También existe la variante llamada direccionamiento indirecto de registro, la cual funciona de la misma manera, pero en vez de utilizar una dirección de memoria, utiliza un registro.

En **direccionamiento indexado** se utiliza un registro índice para almacenar un offset que se le agrega al operando para que como resultado se obtenga la dirección efectiva de los datos.

En el **Direccionamiento de pila** se asume que el operando está en la pila.

También existen el modo de **direccionamiento indirecto indexado**, que usan ambos direccionamientos en simultáneo. Además están los modos de **auto-incremento** y **auto-decremento**, que incrementan o decrementan el registro usado.

Tener varios modos de direccionamiento aumentan el rango de lugares de donde accedemos a datos, generando flexibilidad, pero se pierde simplicidad en el cálculo de direcciones.

Explicar el concepto detrás del pipeline. Conflictos

En el ciclo de instrucción, cada pulso de reloj se utiliza para controlar un paso en la secuencia. Las CPUs actuales dividen este ciclo en pasos más pequeños donde algunos de ellos se pueden realizar en paralelo. Esta superposición acelera la ejecución de los ciclos, ya que en lugar de esperar un ciclo completo para ejecutar el siguiente, varios ciclos se realizan en simultáneo. Este método es el que se conoce como *pipelining*.

La idea es que diferentes pasos completen distintas instrucciones en paralelo donde cada uno de estos pasos se denomina *pipeline stage*.

El objetivo es equilibrar el tiempo que toma cada etapa del pipeline. Si las etapas no están equilibradas, las etapas más rápidas esperan a las más lentas.

Hay que tener en cuenta que no todas las instrucciones deben pasar por cada etapa del pipeline. Por ejemplo, no es lo mismo tener instrucciones con 2 operandos que sin operandos. Sin embargo, para simplificar el hardware del pipeline y la sincronización, todas las instrucciones pasan por todas las etapas, sean o no necesarias.

A partir de lo visto hasta ahora podríamos decir que cuantas más etapas haya en el pipeline, más rápidas serán las ejecuciones. Esto es cierto hasta cierto punto, ya que la cantidad de lógica de control para el pipeline también aumenta de tamaño proporcionalmente al número de etapas, lo que ralentiza la ejecución total. Además, también existen varias condiciones que resultan en conflictos de pipeline que nos impiden alcanzar el objetivo de ejecutar una instrucción por ciclo de reloj:

1. Conflictos de recursos: por ejemplo, si una instrucción requiere almacenar algo en memoria mientras se quiere recuperar otra instrucción de memoria, ambas acciones necesitan acceder a la memoria. En general esto se resuelve permitiendo que continúe la ejecución de la instrucción mientras se obliga a esperar a la búsqueda de la instrucción, o en otros casos se proporcionan dos vías separadas, unas para datos de memoria y otras para instrucciones
2. Dependencia de datos: esto surge cuando el resultado de una instrucción, que todavía no está disponible, debe usarse como operando para una instrucción siguiente. Para resolver este conflicto se puede agregar hardware especial para detectar instrucciones cuyos operandos de origen son destinos de instrucciones más arriba en el pipeline. Este hardware puede insertar un breve retraso (por lo general, una instrucción que no hace nada) en el pipeline, lo que permite que pase suficiente tiempo para resolver el conflicto. También se puede utilizar hardware especializado para detectar estos conflictos y colocar datos a través de rutas especiales que existen entre varias etapas del pipeline. Esto reduce el tiempo necesario para que la instrucción acceda al operando requerido. Algunas arquitecturas abordan este problema dejando que el compilador resuelva el conflicto. Los compiladores reordenan las instrucciones, provocando un retraso en la

carga de cualquier dato en conflicto, pero no tiene ningún efecto sobre la lógica o la salida del programa

3. Saltos condicionales (branching): las instrucciones de bifurcación nos permiten alterar el flujo de ejecución de un programa, que en términos de pipeline, causa problemas importantes. Si las instrucciones se obtienen una por ciclo de reloj, varias de ellas se pueden obtener e incluso decodificar antes de que se ejecute una instrucción anterior, que indica una bifurcación. La ramificación condicional es particularmente difícil de manejar. Muchas arquitecturas ofrecen predicción de bifurcaciones, utilizando la lógica para hacer la mejor estimación de qué instrucciones se necesitarán a continuación. Los compiladores intentan resolver los problemas de bifurcación reorganizando el código de la máquina para provocar una bifurcación retrasada. Se intenta reordenar e insertar instrucciones útiles, y si eso no es posible, se insertan instrucciones no operativas para mantener el pipeline lleno. Otro enfoque utilizado por algunas máquinas es iniciar búsquedas en ambas rutas de la rama y guardarlas hasta que la rama se ejecute realmente, en cuyo momento se conocerá la ruta de ejecución "verdadera".

¿Qué es la memoria RAM? ¿Cuál es más rápida SRAM o DRAM?

Hay dos tipos de memoria: RAM (random access memory) y ROM (read-only memory). La RAM se le suele llamar por "memoria principal", se utiliza para almacenar los programas y los datos que la computadora necesita cuando ejecuta programas. Pero la RAM es volátil y pierde la información cuando se apaga la computadora. Hay dos tipos de chips que se usan para hacer la RAM: SRAM y DRAM (static y dynamic).

La RAM dinámica necesita una recarga cada pocos milisegundos para mantener sus datos. Mientras que la RAM estática mantiene sus contenidos siempre que tenga energía la máquina. Se utiliza a la SRAM para la caché porque es mucho más rápida que la DRAM, pero al ser muy cara, no puede haber mucho de ella. La DRAM se utiliza para la memoria principal, ya que es mucho más densa (almacena muchos bits por chip), usa menos energía, es más barata y genera menos calor que la SRAM.

¿Qué es la memoria ROM y qué tipos existen?

La ROM es la memoria que almacena información crítica para operar el sistema, como el programa necesario para arrancar la computadora. La ROM no es volátil y siempre conserva sus datos. Este tipo de memoria también se usa en sistemas embebidos o en cualquier sistema donde no es necesario cambiar la programación (electrodomésticos, juguetes, la mayoría de los automóviles usan chips ROM, calculadoras, dispositivos periféricos como impresoras láser).

Hay cinco tipos básicos diferentes de ROM: ROM, PROM, EPROM, EEPROM y memoria flash. PROM (programmable read-only memory) es una variación de la ROM. Las PROM pueden ser

programadas por el usuario con el equipo apropiado (tienen fusibles que se pueden quemar para programar el chip), mientras que las ROM están cableadas. Una vez programados, los datos e instrucciones en PROM no se pueden cambiar. Las EPROM (erasable PROM) son programables con la ventaja adicional de ser reprogramables (borrar una EPROM requiere una herramienta especial que emite luz ultravioleta). Para reprogramar una EPROM, primero se debe borrar todo el chip. Las EEPROM (electrically erasable PROM) elimina muchas de las desventajas de EPROM: no se requieren herramientas especiales para el borrado (esto se realiza aplicando un campo eléctrico) y solo puede borrar partes del chip, un byte a la vez. La memoria flash es esencialmente EEPROM con el beneficio adicional de que los datos se pueden escribir o borrar en bloques, eliminando la limitación de un byte a la vez. Esto hace que la memoria flash sea más rápida que la EEPROM.

Jerarquía de la memoria

No todas las memorias son iguales y algunas son mucho menos eficientes que otras (y por lo tanto, más baratas). Para hacer frente a esta disparidad, los sistemas informáticos actuales utilizan una combinación de tipos de memoria para proporcionar el mejor rendimiento al mejor costo. Este enfoque se llama memoria jerárquica. Cuanto más rápida es la memoria, más cara es por bit de almacenamiento. Mediante el uso de una jerarquía de memorias, cada una con diferentes velocidades de acceso y capacidades de almacenamiento, un sistema informático puede exhibir un rendimiento superior al que sería posible sin una combinación de los distintos tipos (rendimiento aceptable a un costo razonable). Los tipos básicos que normalmente constituyen el sistema de memoria jerárquico incluyen registros, caché, memoria principal y memoria secundaria.

La memoria se suele clasificar en función de su "distancia" desde el procesador, con la distancia medida por la cantidad de ciclos de máquina necesarios para el acceso. Cuanto más cerca esté la memoria del procesador, más rápida debería ser. Por lo tanto, se utilizan tecnologías más lentas para las memorias más lejanas y tecnologías más rápidas para memorias más cercanas a la CPU. Cuanto mejor es la tecnología, más rápida y costosa se vuelve la memoria. Es por eso que las memorias más rápidas tienden a ser más pequeñas que las más lentas, debido al costo.

Existe cierta terminología que se usa para referirse a las memorias:

- Hit: los datos solicitados residen en alguno de los niveles de la memoria (en general, se refiere a los niveles superiores)
- Miss: los datos solicitados no se encuentran en el nivel de memoria dado
- Hit rate: el porcentaje de accesos encontrados en un nivel determinado
- Miss rate: el porcentaje de accesos no encontrados en un nivel determinado. $\text{Miss rate} = 1 - \text{Hit rate}$
- Hit time: el tiempo necesario para acceder a la información solicitada en un nivel determinado
- Miss penalty: el tiempo requerido para procesar un error, que incluye reemplazar un bloque en un nivel superior de memoria, más el tiempo adicional para entregar los datos

solicitados al procesador (el tiempo para procesar un fallo suele ser superior al tiempo para procesar un acierto)

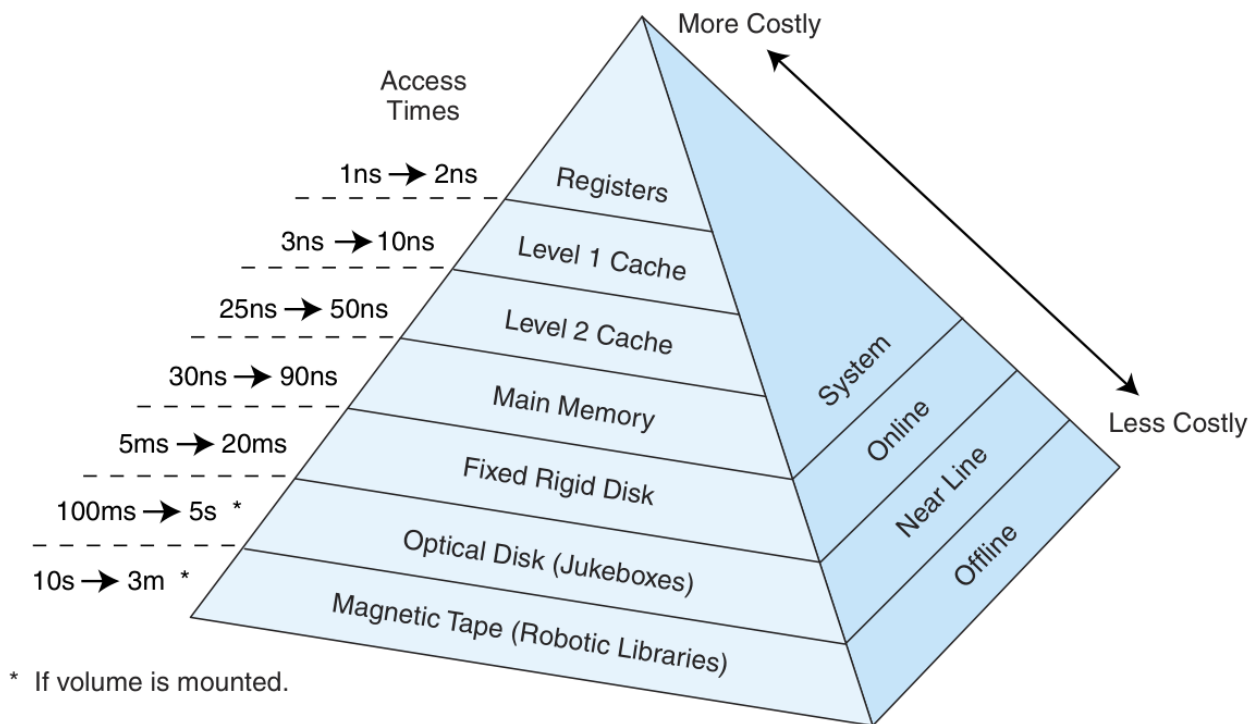


FIGURE 6.1 The Memory Hierarchy

Explicar el concepto de “locality of reference” y sus tres tipos

Cuando se procesa un miss, en vez de simplemente transferir el dato solicitado a un nivel más alto (de la jerarquía de la memoria), el bloque entero que contiene los datos se transfiere. Es probable que los datos adicionales que incluye el bloque sean utilizados en un futuro cercano al pedido del dato original, y de esa forma se pueden cargar esos datos desde una memoria más rápida.

Hay tres formas de localidad:

- Localidad temporal: los ítems accedidos recientemente tienden a ser accedidos de vuelta en el futuro cercano
- Localidad espacial: los accesos tienden a agruparse en el espacio de la dirección (por ejemplo, en los arreglos o ciclos)
- Localidad secuencial: las instrucciones tienden a accederse secuencialmente

El principio de localidad le permite a un sistema usar una pequeña cantidad de memoria de alta velocidad para efectivamente acelerar la mayoría de los accesos a memoria. Normalmente, se accede muchas veces a una pequeña cantidad de memoria, la cual se copia en uno de los niveles más altos de la jerarquía.

Esto resulta en tener mucha información en una memoria de bajo costo y lenta, manteniendo buena velocidad de acceso.

¿Qué es la memoria caché y para qué sirve?

El procesador de una computadora es muy rápido y está constantemente leyendo información de la memoria, lo que significa que a menudo tiene que esperar a que llegue la información, porque los tiempos de acceso a la memoria son más lentos que la velocidad del procesador. Una memoria caché es una memoria pequeña (cercana a la CPU), temporal, pero rápida que el procesador utiliza para la información que probablemente necesitará nuevamente en un futuro muy cercano. Almacena datos a los que se ha accedido y datos a los que la CPU podría acceder próximamente.

Como la computadora no tiene forma de saber a priori qué datos son más probables que se accedan, se utiliza el principio de localidad para transferir los bloques a caché siempre que se haga un acceso a la memoria principal. Si la probabilidad de usar algo más en ese bloque es alta, la transferencia de todo el bloque ahorra tiempo de acceso. La ubicación de caché para este nuevo bloque depende de dos cosas: la política de mapeo de caché y el tamaño de caché (que afecta si hay espacio para el nuevo bloque).

El propósito de la memoria caché es acelerar los accesos a la memoria al almacenar los datos usados recientemente más cerca de la CPU, en lugar de almacenarlos en la memoria principal. Aunque la caché no es tan grande como la memoria principal, es considerablemente más rápida. La caché no necesita ser muy grande para funcionar bien. Una regla general es hacer que la caché sea lo suficientemente pequeña para que el costo promedio general por bit sea cercano al de la memoria principal, pero lo suficientemente grande como para ser beneficioso. Debido a que esta memoria rápida es bastante costosa, no es factible usar la tecnología que se encuentra en la memoria caché para construir toda la memoria principal.

Diferencias entre caché L1 y caché L2

El tamaño de la memoria caché puede variar enormemente. El caché de nivel 2 (L2) de una computadora personal típica es de 256K o 512K. El caché de nivel 1 (L1) es más pequeño, normalmente de 8K o 16K. La caché L1 reside en el procesador, mientras que la caché L2 reside entre la CPU y la memoria principal. La caché L1 es, por lo tanto, más rápida que la caché L2. La relación entre la memoria caché L1 y L2 se puede ilustrar usando un ejemplo: si un supermercado fuera la memoria principal, la heladera sería la memoria caché L2 y la mesa real la memoria caché L1.

Esquemas de mapeo de caché

A la memoria caché no se accede por dirección, se accede por contenido. Para que la caché sea funcional, debe almacenar datos útiles. Sin embargo, estos datos se vuelven inútiles si la

CPU no puede encontrarlos. Al acceder a datos o instrucciones, la CPU primero genera una dirección de memoria principal. Si los datos se han copiado en la memoria caché, la dirección de los datos en la memoria caché no es la misma que la dirección de la memoria principal. Para simplificar este proceso de ubicar los datos deseados, se utilizan varios algoritmos de mapeo de caché (algo así como convertir la dirección de la memoria principal en una dirección de caché). En la mayoría de los esquemas de mapeo de caché, las entradas de caché deben verificarse o buscarse para ver si el valor que se solicita está almacenado en caché. Esta conversión de direcciones se realiza otorgando un significado especial a los bits en la dirección de la memoria principal. Se dividen los bits en distintos grupos que llamamos campos. Dependiendo del esquema de mapeo, podemos tener dos o tres campos que se usan de distinta forma dependiendo del esquema a seguir. El esquema de mapeo determina dónde se colocan los datos cuando se copian originalmente en la memoria caché y también proporciona un método para que la CPU encuentre datos copiados previamente cuando busca en la memoria caché.

La memoria principal y la memoria caché se dividen en bloques del mismo tamaño (el tamaño de estos bloques varía). Cuando se genera una dirección de memoria, primero se busca en el caché para ver si la palabra requerida existe allí. Cuando la palabra solicitada no se encuentra en la memoria caché, todo el bloque de memoria principal en el que reside la palabra se carga en la memoria caché. Este esquema tiene éxito debido al principio de localidad. Una palabra perdida puede generar como resultado varias palabras encontradas.

Un campo de la dirección de la memoria principal nos señala una ubicación en la memoria caché en la que residen los datos si residen en la memoria caché (hit), o donde se colocarán si no son residentes (miss). El bloque de memoria caché al que se hace referencia se verifica luego para ver si es válido. Esto se hace asociando un bit válido con cada bloque de caché. Un bit en 0 significa que el bloque de caché no es válido (miss) y debemos acceder a la memoria principal. Un bit en 1 significa que es válido (podemos tener un hit, pero todavía no es seguro). Luego comparamos la etiqueta en el bloque de caché con el campo de etiqueta de nuestra dirección (la etiqueta es un grupo especial de bits derivados de la dirección de memoria principal que se almacena con su bloque correspondiente en la memoria caché). Si las etiquetas son las mismas, hemos encontrado el bloque de memoria caché deseado. Luego ubicamos la palabra deseada en el bloque, usando una parte diferente de la dirección de la memoria principal llamada campo de palabra. Todos los esquemas de mapeo de caché requieren un campo de palabra; sin embargo, los campos restantes están determinados por el esquema de mapeo.

Tipos de esquemas

Caché de mapeo directo: debido a que hay más bloques de memoria principal que de caché, los bloques de memoria principal compiten por posiciones en la caché. Este mapeo consiste en asignar el bloque X de memoria principal al bloque Y de caché $\text{mod } N$, donde N es el número total de bloques de caché. Cada bloque copiado se identifica con una etiqueta.

La dirección de memoria principal se divide en 3 campos que suman en total el número de bits de una dirección de memoria principal: uno de los campos indica la etiqueta, otro el bloque y

otro la palabra. El tamaño de cada campo depende de las características físicas de la memoria principal y la caché. El campo de palabra identifica de forma única una palabra de un bloque específico; por lo tanto, debe contener el número apropiado de bits para hacer esto. Esto también se aplica al campo de bloque: debe seleccionar un bloque único de caché. El campo de la etiqueta es lo que sobra. Cuando un bloque de memoria principal se copia a la memoria caché, esta etiqueta se almacena con el bloque y lo identifica de manera única. Si un bloque ya ocupa la ubicación de caché donde se debe colocar un nuevo bloque, se elimina el bloque actual (se vuelve a escribir en la memoria principal si se ha modificado o simplemente se sobrescribe si no se ha cambiado) y se coloca el nuevo. Este tipo de memoria no es tan costosa como otras porque el esquema de asignación no requiere ninguna búsqueda.

Main Memory	Maps To	Cache
Block 0 (addresses 0, 1)	→	Block 0
Block 1 (addresses 2, 3)	→	Block 1
Block 2 (addresses 4, 5)	→	Block 2
Block 3 (addresses 6, 7)	→	Block 3
Block 4 (addresses 8, 9)	→	Block 0
Block 5 (addresses 10, 11)	→	Block 1
Block 6 (addresses 12, 13)	→	Block 2
Block 7 (addresses 14, 15)	→	Block 3

TABLE 6.1 An Example of Main Memory Mapped to Cache

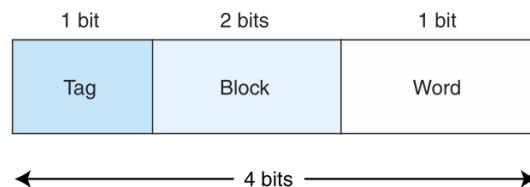


FIGURE 6.6 The Main Memory Address Format for a 16-Word Memory

Caché completamente asociativa: esta memoria se basa en la idea de que cada bloque de memoria principal se pueda colocar en cualquier lugar de la memoria caché. La única forma de encontrar un bloque mapeado de esta manera es buscar en todo el caché: el campo de etiqueta de la dirección de memoria principal se compara con todos los campos de etiqueta válidos en la memoria caché; si se encuentra una coincidencia, se encuentra el bloque. Si no hay coincidencia, tenemos un miss de caché y el bloque debe ser transferido desde la memoria principal.

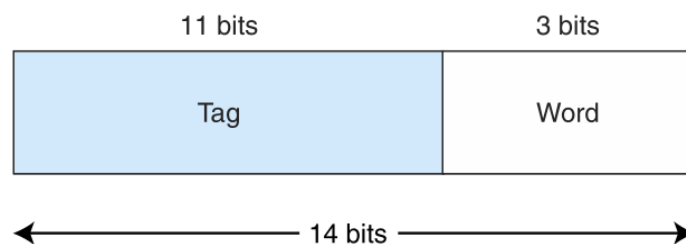


FIGURE 6.8 The Main Memory Address Format for Associative Mapping

La memoria asociativa requiere un hardware especial para permitir la búsqueda por etiquetas, y es, por lo tanto, más cara. En este caso la dirección de la memoria principal se divide en dos partes, la etiqueta y la palabra.

Cuando la memoria caché está llena, necesitamos un algoritmo de reemplazo para decidir qué bloque deseamos eliminar de la memoria caché (victim block). Por ejemplo, uno sencillo sería first in-first out (FIFO).

Asociativa por conjuntos de n-vías: Este esquema es similar al mapeo de caché directo, en el que usamos la dirección para mapear el bloque a una determinada ubicación de caché. La diferencia importante es que en lugar de mapear a un solo bloque de caché, una dirección se asigna a un conjunto de varios bloques de caché. Todos los conjuntos en caché deben ser del mismo tamaño. Por ejemplo, en una caché asociativa por conjuntos de dos vías, hay dos bloques por conjuntos.

Set	Tag	Block 0 of set	Valid	Tag	Block 1 of set	Valid
0	00000000	Words A, B, C, ...	1	-----		0
1	11110101	Words L, M, N, ...	1	-----		0
2	-----		0	10111011	P, Q, R, ...	1
3	-----		0	11111100	T, U, V, ...	1

FIGURE 6.9 A Two-Way Set Associative Cache

Se puede decir que el caché de mapeo directo es un caso especial del asociativo por conjuntos, siendo de 1 vía (es decir, el tamaño del conjunto es uno).

En este esquema, la dirección a memoria principal se particiona en tres: el campo de etiqueta, el campo de conjunto (set) y el campo de la palabra. Los campos de la etiqueta y de la palabra ocupan los mismos roles; el campo del conjunto indica en qué conjunto de caché el bloque de memoria principal mapea. Ejemplo para caché de 16 bloques, 8 sets (2 bloques por set), 8 palabras por bloque.

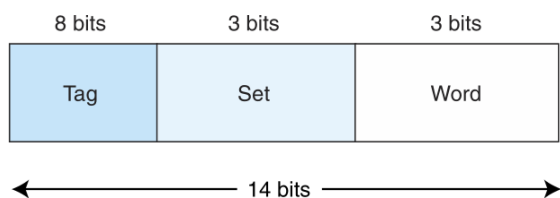


FIGURE 6.10 Format for Set Associative Mapping

Políticas de reemplazo

En el mapeo directo, si hay una disputa simplemente se elimina el bloque que está en memoria y se coloca el nuevo, debido a que la ubicación de cada bloque está predeterminada.

Tanto en caché totalmente asociativa como en caché asociativa por conjuntos es necesario utilizar algoritmos para reemplazar los bloques. A estos algoritmos se los denomina políticas de

reemplazo y utilizan un bloque víctima que es el que se va a eliminar cuando se llena la memoria. Cuando se utiliza una memoria caché totalmente asociativa, hay K posibles ubicaciones de memoria caché (donde K es el número de bloques en la memoria caché) a las que se puede asignar un bloque de memoria principal determinado. Con el mapeo asociativo de conjuntos de N-vías, un bloque puede mapear a cualquiera de N bloques diferentes dentro de un conjunto dado.

Hay varias políticas de reemplazo populares. Uno que no es práctico pero que puede usarse como punto de referencia para medir todos los demás es el algoritmo óptimo. La idea sería mantener los valores en la memoria caché que se necesitarán de nuevo pronto y descartar los bloques que no se usarán durante algún tiempo o que ya no sean necesarios. Debido a que no podemos ver el futuro en cada programa que ejecutamos, el algoritmo óptimo se usa solo como una métrica para determinar qué tan bueno o malo es otro algoritmo. Cuanto más se acerque un algoritmo al óptimo, mejor.

Una posible política de reemplazo considera la localidad temporal. Cualquier valor que no se haya utilizado recientemente es poco probable que se use en un corto plazo. Se realiza un seguimiento de la última vez que se accedió a cada bloque (se asigna un tiempo) y se selecciona como bloque víctima el bloque que se ha utilizado menos recientemente. Este es el algoritmo *least recently used* (LRU). Desafortunadamente, LRU requiere que el sistema mantenga un historial de accesos para cada bloque de caché, lo que requiere un espacio significativo que ralentiza el funcionamiento de la caché.

Otro algoritmo posible es FIFO (first in first out). El bloque que estuvo en la caché durante más tiempo (independientemente de la última vez que se haya utilizado) se seleccionaría como la víctima a eliminar.

Otro enfoque es seleccionar una víctima al azar. El problema con LRU y FIFO es que hay situaciones en las que genera *thrash* (quitar constantemente un bloque y traerlo de vuelta, repetidamente). Aunque el reemplazo aleatorio a veces arroja datos que se necesitarán pronto, nunca genera *thrash*. Desafortunadamente, es difícil tener un reemplazo verdaderamente aleatorio y se puede disminuir el rendimiento promedio.

El algoritmo seleccionado a menudo depende de cómo se utilizará el sistema. Ningún algoritmo único es mejor para todos los escenarios.

Tiempo de acceso efectivo y tasa de aciertos

El rendimiento de una memoria jerárquica se mide por su tiempo de acceso efectivo (EAT), o el tiempo promedio por acceso. EAT es un promedio ponderado que utiliza la tasa de aciertos y los tiempos de acceso relativos de los sucesivos niveles de la jerarquía. Por ejemplo, para una memoria de dos niveles sería:

$$\text{EAT} = H \times \text{AccesoC} + (1 - H) \times \text{AccesoMM}$$

Donde H = tasa de aciertos de caché (hit rate), AccesoC = tiempo de acceso a la memoria caché, y AccesoMM = tiempo de acceso a la memoria principal

Políticas de escritura de caché

Se llama *dirty blocks* (bloques sucios) a los bloques de caché que se modifican. Cuando el procesador escribe en la memoria principal, los datos pueden escribirse en el caché en su lugar, suponiendo que el procesador probablemente los volverá a leer pronto. Si se modifica un bloque de caché, la política de escritura de caché determina cuándo se actualiza el bloque de memoria principal real para que coincida con el bloque de caché. Hay dos políticas básicas de escritura:

Escritura simultánea (Write-through): una política de escritura simultánea actualiza tanto la memoria caché como la memoria principal simultáneamente en cada escritura. Esto es más lento que la reescritura (pues cada escritura requiere acceso a la memoria principal), pero garantiza que la memoria caché sea consistente con la memoria principal del sistema. En general, en aplicaciones reales, la mayoría de los accesos son de lectura, por lo que esta ralentización es despreciable.

Reescritura (Write-back): una política de reescritura solo actualiza bloques en la memoria principal cuando el bloque de caché se selecciona como víctima y debe eliminarse de la memoria caché. Normalmente, esto es más rápido que la escritura simultánea porque no se pierde tiempo escribiendo información en la memoria principal en cada escritura de caché. El tráfico de memoria también se reduce. La desventaja es que la memoria principal y la memoria caché pueden no contener el mismo valor en un instante de tiempo dado, y si un proceso termina (falla) antes de que se complete la escritura en la memoria principal, los datos en la memoria caché pueden perderse.

Para mejorar el rendimiento de la memoria caché, se debe aumentar la proporción de aciertos mediante el uso de un mejor algoritmo de mapeo (hasta aproximadamente un aumento del 20 %), mejores estrategias para operaciones de escritura (potencialmente un aumento del 15 %), mejores algoritmos de reemplazo (hasta un aumento del 10 %) y mejores prácticas de codificación (hasta un aumento del 30 % en la tasa de hit). El simple aumento del tamaño de la memoria caché puede mejorar la tasa de aciertos aproximadamente entre un 1% y un 4%, pero no se garantiza que así sea.

Arquitectura de entrada y salida

Definiremos entrada/salida como un subsistema de componentes que mueve datos codificados entre dispositivos externos y un sistema host, que consta de una CPU y una memoria principal. Los subsistemas de E/S incluyen, pero no se limitan a:

- Bloques de memoria principal que se dedican a funciones de E/S
- Buses que proporcionan los medios para mover datos dentro y fuera del sistema
- Módulos de control en el host y en dispositivos periféricos
- Interfaces a componentes externos como teclados y discos
- Cableado o enlaces de comunicaciones entre el sistema host y sus periféricos

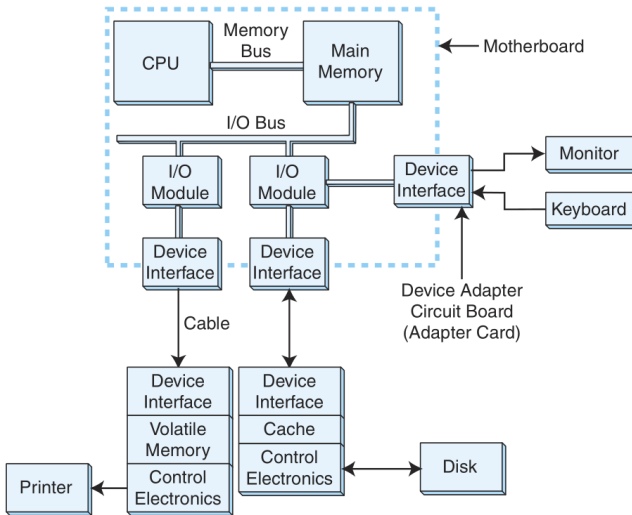


FIGURE 7.1 A Model I/O Configuration

Los módulos de E/S se encargan de mover datos entre la memoria principal y una interfaz de un dispositivo en particular. Las interfaces están diseñadas específicamente para comunicarse con ciertos tipos de dispositivos, como teclados, discos o impresoras. Las interfaces manejan los detalles de asegurarse de que los dispositivos estén listos para la siguiente tanda de datos, o que el host esté listo para recibir la siguiente tanda de datos proveniente del dispositivo periférico.

La forma exacta y el significado de las señales intercambiadas entre un emisor y un receptor se denomina *protocolo*. Los protocolos consisten de señales de comando, como "Reinicio de la impresora"; señales de estado, como "Cinta lista"; o señales de paso de datos, como "Aquí están los bytes que solicitó". En la mayoría de los protocolos de intercambio de datos, el receptor debe reconocer los comandos y los datos que se le envían o indicar que está listo para recibir datos. Este tipo de intercambio de protocolo se denomina *handshake* (apretón de manos).

Muchos buses de E/S reales no tienen direcciones y líneas de datos separadas. Debido a la naturaleza asíncrona de un bus de E/S, las líneas de datos se pueden utilizar para almacenar la dirección del dispositivo. Todo lo que necesitamos hacer es agregar otra línea de control que indique si las señales en las líneas de datos representan una dirección o un dato.

Explique los cuatro métodos de control de entrada-salida (I/O)

I/O = E/S

Los sistemas informáticos emplean cualquiera de los cuatro métodos generales de control de E/S. Aunque un método no es necesariamente mejor que otro, la forma en que una computadora controla su E/S influye en gran medida en el diseño y el rendimiento general del sistema. El objetivo es saber cuándo el método usado es apropiado para cómo se usará el sistema.

Los sistemas que utilizan **I/O programadas** dedican al menos un registro para el uso exclusivo de cada dispositivo de E/S. La CPU monitorea continuamente cada registro, esperando que lleguen los datos. Esto se llama *polling*. Una vez que la CPU detecta una condición de "datos listos", actúa de acuerdo a las instrucciones programadas para ese registro en particular.

El beneficio de usar este enfoque es que tenemos control programático sobre el comportamiento de cada dispositivo. Los cambios de programa pueden hacer ajustes al número y tipos de dispositivos en el sistema, así como a sus prioridades e intervalos de polling. Sin embargo, el polling (sondeo) constante de registros es un problema. La CPU está en continuo bucle de "espera ocupada" hasta que comienza a atender una solicitud de E/S. No hace ningún trabajo útil hasta que haya una E/S para procesar. Debido a estas limitaciones, la E/S programada se adecua más a sistemas de propósito especial como cajeros automáticos y sistemas que controlan o monitorean eventos ambientales.

En la **E/S controlada por interrupción**, en lugar de que la CPU continuamente pregunte a sus dispositivos si hay alguna entrada, los dispositivos le avisan a la CPU cuando tienen datos para enviar. La CPU procede con otras tareas hasta que un dispositivo que solicita el servicio lo interrumpe. Las interrupciones generalmente se señalan con un bit en el registro de flags de la CPU llamado flag de interrupción.

Una vez que se establece el indicador de interrupción, el sistema operativo interrumpe cualquier programa que se esté ejecutando actualmente, guardando el estado de ese programa y la información variable. Luego, el sistema obtiene el *vector de dirección* que apunta a la dirección de la rutina de E/S. Una vez que la CPU ha completado el servicio de E/S, restaura la información que guardó del programa que se estaba ejecutando cuando ocurrió la interrupción, y se reanuda la ejecución del programa.

La E/S impulsada por interrupciones es similar a la E/S programada en que las rutinas de servicio se pueden modificar para adaptarse a los cambios de hardware.

Cuando un sistema usa **acceso directo a memoria** (DMA), la CPU se desprende de la ejecución de tediosas instrucciones de E/S. Para efectuar la transferencia, la CPU proporciona al controlador DMA la ubicación de los bytes que se transferirán, el número de bytes que se

transferirán y el dispositivo de destino o dirección de memoria. Esta comunicación suele tener lugar a través de registros de E/S especiales en la CPU.

Una vez que los valores adecuados se colocan en la memoria, la CPU envía una señal al subsistema DMA y continúa con su siguiente tarea, mientras que el DMA se ocupa de los detalles de la E/S. Una vez que se completa la E/S (o termina con un error), el subsistema DMA envía una señal a la CPU enviándole otra interrupción.

El controlador DMA y la CPU comparten el bus de memoria y solo uno de ellos a la vez puede tener el control del mismo (bus master). En general, la E/S tiene prioridad sobre las búsquedas en la memoria de la CPU porque muchos dispositivos de E/S funcionan dentro de parámetros de tiempo ajustados. Para evitar los tiempos de espera del dispositivo, el DMA utiliza ciclos de memoria que, de otro modo, utilizaría la CPU. Afortunadamente, la E/S tiende a crear tráfico en ráfagas en el bus: los datos se envían en bloques o grupos. La CPU debe tener acceso al bus entre ráfagas, aunque es posible que este acceso no tenga la duración suficiente para evitar que el sistema se acuse de "rastrear durante la E/S".

Este es justamente el problema que tiene el DMA: compite con la CPU accediendo muchas veces a memoria e impide su funcionamiento normal.

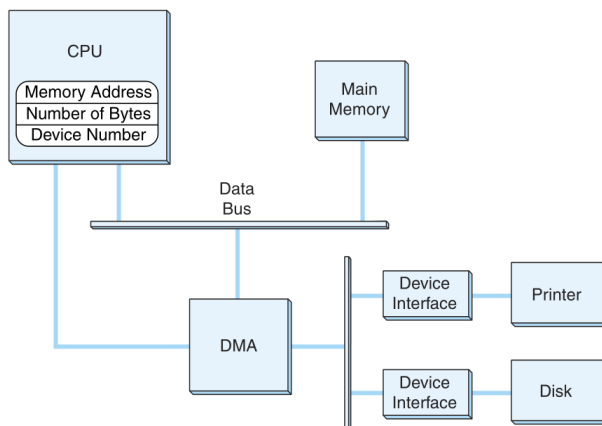


FIGURE 7.2 An Example DMA Configuration

La **E/S de canal** es un tipo inteligente de interfaz DMA que sirve para sistemas grandes de múltiples usuarios, como computadoras centrales. Uno o más procesadores de E/S controlan varias rutas de E/S denominadas rutas de canal. Las rutas de canales para dispositivos "lentos", como terminales e impresoras, se pueden combinar (multiplexar), lo que permite la gestión de varios de estos dispositivos a través de un solo controlador. Los canales para unidades de disco y otros dispositivos "rápidos" se denominan canales selectores.

Los canales de E/S están controlados por pequeñas CPU denominadas procesadores de E/S (IOP), que están optimizadas para E/S. A diferencia de los circuitos DMA, los IOP tienen la capacidad de ejecutar programas que incluyen instrucciones aritmético-lógicas y de ramificación. Los IOP ejecutan programas que el procesador principal coloca en la memoria principal del sistema. Estos programas incluyen no solo las instrucciones de transferencia reales, sino también comandos que controlan los dispositivos de E/S. Una vez que el programa de E/S se ha colocado en la memoria, el host informa al IOP la ubicación del mismo. El IOP

luego completa su trabajo, coloca la información de finalización en la memoria y envía una interrupción a la CPU. La CPU obtiene la información de finalización y toma las medidas apropiadas para los códigos de retorno.

La distinción principal entre DMA y E/S de canal radica en la inteligencia del IOP. El host solo tiene que crear las instrucciones del programa para la operación de E/S y decirle al IOP dónde encontrarlas.

Un IOP debe robar ciclos de memoria de la CPU. Los sistemas de E/S de canal están equipados con buses de E/S separados, que ayudan a aislar el host de la operación de E/S. El IOP usa el bus de memoria del sistema solo para obtener sus instrucciones de la memoria principal. El resto de la transferencia se realiza utilizando solo el bus de E/S. Debido a su inteligencia y aislamiento de bus, la E/S de canal se utiliza en entornos de procesamiento de transacciones de alto rendimiento, donde su costo y complejidad pueden justificarse.

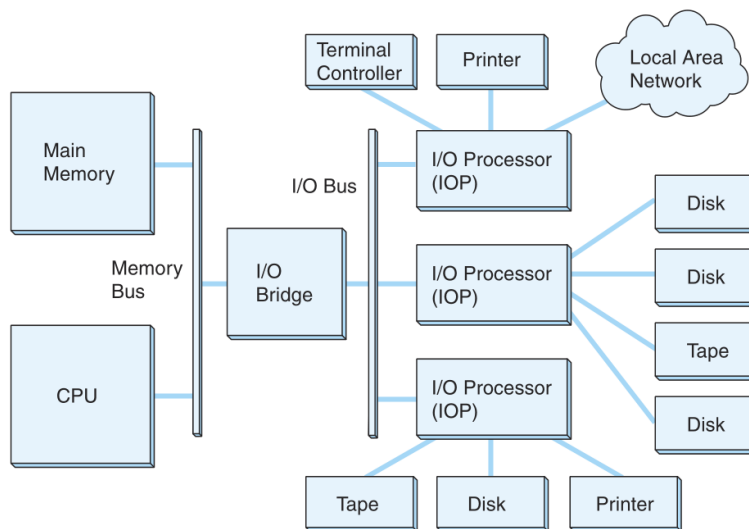


FIGURE 7.3 A Channel I/O Configuration

Interrupciones en sistemas pequeños

Los periféricos cuentan con una forma de comunicarse con la CPU en lugar de permanecer inactivo hasta que llega un comando para hacer lo contrario. Cada dispositivo periférico del sistema tiene acceso a una línea de solicitud de interrupción. El chip de control de interrupción tiene una entrada para cada línea de interrupción. Cada vez que hay una interrupción, el controlador decodifica la interrupción y genera la entrada de interrupción (INT) en la CPU. Cuando la CPU está lista para procesar la interrupción, afirma la señal de reconocimiento de interrupción (INTA). Una vez que el controlador de interrupciones recibe este reconocimiento, puede bajar su señal INT.

Los diseñadores de sistemas deben, por supuesto, decidir qué dispositivos deben tener prioridad sobre los demás cuando más de un dispositivo genera interrupciones

simultáneamente. El número de líneas de solicitud de interrupción están limitadas en todos los sistemas y, en algunos casos, la interrupción se puede compartir. Las interrupciones compartidas no causan problemas cuando está claro que dos dispositivos no necesitarán la misma interrupción al mismo tiempo.

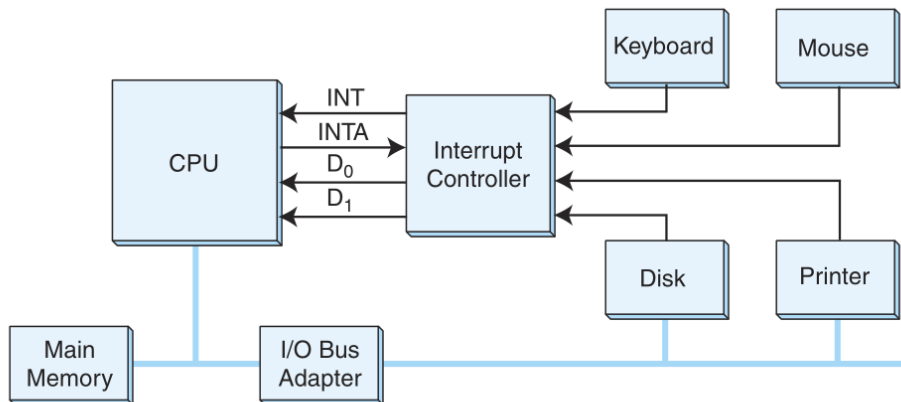


FIGURE 7.8 An I/O Subsystem Using Interrupts

¿Cómo se puede mejorar el rendimiento de una computadora?

Para mejorar el rendimiento general de una computadora se pueden realizar estas acciones:

- Optimización de la CPU - Maximiza la velocidad y eficiencia de las operaciones realizadas por la CPU.
- Optimización de la Memoria - Maximiza la eficiencia de la administración de memoria de un código.
- Optimización de E/S - Maximiza la eficiencia de las operaciones de E/S.

Una aplicación cuyo rendimiento esté limitado por una de estas se les llama: CPU bound (ligado), memory bound, I/O bound.

Benchmark

Performance benchmarking es la ciencia de hacer evaluaciones objetivas del desempeño de un sistema sobre otro. Los benchmarks también son útiles para evaluar las mejoras de rendimiento obtenidas al actualizar una computadora o sus componentes. Los buenos benchmarks nos permiten superar el hype publicitario y los trucos estadísticos. En última instancia, los buenos benchmarks identificarán los sistemas que brindan un buen rendimiento al costo más razonable.

Velocidad de reloj, MIPS y FLOPS

La velocidad de la CPU es una métrica engañosa. En sistemas arquitectónicamente idénticos, es probable que una CPU que funciona al doble de la velocidad de reloj que otra proporcione un mejor rendimiento de la CPU. Pero al comparar distintos sistemas, es probable que no sean idénticos desde el punto de vista arquitectónico.

Una métrica ampliamente citada relacionada con la frecuencia del reloj es la métrica de millones de instrucciones por segundo (MIPS). Esto mide la velocidad a la que el sistema puede ejecutar una combinación típica de instrucciones aritméticas de números enteros y punto flotante, así como operaciones lógicas. La debilidad de esta medida está relacionada a que distintas arquitecturas requieren distinta cantidad de ciclos de máquina para realizar cierta tarea. No se tiene en cuenta la cantidad de instrucciones necesarias para completar una tarea específica.

Otra métrica utilizada es la métrica FLOPS (floating-point operations per second). En esta métrica no existe un acuerdo sobre lo que constituye una operación de punto flotante. Es decir, no hay un acuerdo sobre si tomar como operaciones de punto flotante las operaciones intermedias de una operación mayor, por ejemplo. Por estas razones, se perjudica a los algoritmos que tienen un rendimiento de algoritmos eficientes.

En general MFLOPS (mega-flops) y MIPS son métricas populares entre el marketing porque suena como un valor “sólido” y representan un concepto simple e intuitivo. Es por eso que, a pesar de sus deficiencias, estas métricas se utilizan para comparar el rendimiento relativo en una línea de computadoras similares ofrecidas por el mismo proveedor.

¿Qué son los benchmarks sintéticos? Dé ejemplos

Los benchmarks sintéticos están especialmente diseñados para medir el rendimiento de un componente individual de un ordenador, normalmente llevando el componente escogido a su máxima capacidad.

Ejemplos:

Whetstone (1977): intensivo en operaciones de punto flotante, con llamados a rutinas trigonométricas y exponenciales

Linpack (1984): LINear algebra PACKage. Resuelve sistemas de ecuaciones lineales usando aritmética de doble precisión real. Una de las cosas buenas de este benchmark es que establece una medida estándar para FLOPS. Un sistema que no tiene circuitos de punto flotante puede obtener una calificación FLOPS si lleva a cabo correctamente el benchmark Linpack.

Dhrystone (1984): operaciones de strings y enteros.

La ventaja que tienen estos benchmarks es que son simples y fáciles de entender.

Desafortunadamente, esa también es su mayor limitante. Es muy fácil para los escritores de compiladores equipar sus productos con interruptores en el compilador que invoquen código

especial que está optimizado para estos benchmarks. Los objetos a compilar son tan chicos que la mayor parte del programa se mantiene en caché, por lo que no se evalúa el manejo de memoria del sistema.

¿Qué son los benchmarks de SPEC?

SPEC es una corporación cuyo principal objetivo es establecer métodos equitativos y realistas para medir el rendimiento de las computadoras.

El más conocido de los benchmark de SPEC mide el rendimiento de la CPU, la velocidad de acceso a la caché y la memoria, y la eficiencia del compilador. La última versión de este benchmark es CPU2000 que consta de dos partes: CINT2000, que mide qué tan bien un sistema realiza el procesamiento de enteros, y CFP2000, que mide el rendimiento de punto flotante.

Aunque lo que se suele querer mejorar es el rendimiento de los procesadores, la ley de Amdahl enuncia que un sistema útil requiere más que una CPU rápida. Debido a que se quiere saber qué tan bien funcionará un sistema completo bajo sus cargas de trabajo particulares, SPEC ha creado una variedad de otras métricas, que incluyen SPEC Web para servidores web, SPEC HPC para computadoras de alto rendimiento y SPEC JVM para el rendimiento de Java del lado del cliente. Cada uno de estos benchmark se adhiere a la filosofía de SPEC de establecer medidas de rendimiento del sistema justas y objetivas.

La corporación tiene tres comitativas, cada una centrada en:

- computadoras de escritorio,
- computadoras de nivel empresarial y supercomputadoras,
- Computadoras que se centran en multimedia y gráficos.

En un principio SPEC había prohibido el uso de los modos especiales de compilador para optimizar el benchmark, pero luego de críticas por parte de la comunidad de vendedores, se decidió establecer un sistema doble. Por un lado se darán los resultados del benchmark con un modo de compilador único para todo el conjunto (suite) y por otro se darán los resultados obtenidos con ajustes optimizados.

¿Qué son los benchmarks de TPC?

Los benchmarks de TPC están relacionados a los compradores de servidores de procesamiento de transacciones empresariales. Para los sistemas de esta clase, los compradores están más interesados en la capacidad del servidor para procesar una gran cantidad de actividades simultáneas de corta duración, donde cada transacción involucra comunicaciones y E/S de disco hasta cierta medida. Los sistemas lentos de procesamiento de transacciones pueden ser enormemente costosos para las empresas y causar problemas de gran alcance. Con tanto en juego, es esencial encontrar algún método para evaluar objetivamente el rendimiento general de los sistemas que respaldan estos procesos comerciales críticos.

¿Qué es la simulación de sistemas?

En general, las simulaciones nos brindan herramientas que podemos usar para modelar y predecir aspectos de comportamiento del sistema sin el uso del entorno en vivo exacto que el simulador está modelando.

La simulación es muy útil para estimar el rendimiento de sistemas o configuraciones de sistemas que aún no existen. Los diseñadores de sistemas prudentes siempre realizan estudios de simulación de nuevo hardware y software antes de la construcción de versiones comerciales de sus productos.

Las simulaciones son modelos de aspectos particulares de sistemas completos. Brindan a los diseñadores de sistemas el lujo de realizar pruebas hipotéticas en un entorno controlado separado del sistema en vivo.

Los seguimientos del sistema (system traces) recopilan información de comportamiento detallada utilizando sondas de hardware o software en la actividad del componente de interés. Las sondas rastrean cada detalle del comportamiento real del componente, posiblemente incluyendo instrucciones binarias y referencias de memoria. Los seguimientos recopilados por las sondas consisten en solo unos segundos de actividad del sistema porque la salida del conjunto de datos es muy grande. Para producir un modelo estadísticamente significativo, se requieren varias trazas.

Optimización del rendimiento de la CPU

No existe una forma única de mejorar el rendimiento de la CPU, porque este se ve afectado por una multitud de factores. Por ejemplo, el código del programa afecta el conteo de instrucciones; el compilador influye tanto en el recuento de instrucciones como en los ciclos de reloj promedio por instrucción; el ISA determina el conteo de instrucciones y los ciclos de reloj promedio por instrucción; y la organización real del hardware establece los ciclos de reloj por instrucción y el tiempo de ciclo de reloj. Las posibles técnicas de optimización de la CPU incluyen unidades integradas de punto flotante, unidades de ejecución paralelas, instrucciones especializadas, pipeline de instrucciones, predicción de bifurcaciones y optimización de código.

Optimización de branch

Entre el 20% y 30% de las instrucciones de máquina implican ramificaciones y más o menos el 65% de esas ramificaciones son tomadas. Muchas de ellas detienen los pipelines, ya que no permiten un flujo secuencial del programa. Es por eso que es necesario realizar cierta optimización para que no se detengan frecuentemente estos procesos.

Uno de los métodos utilizados es la **ramificación retardada**. Al realizar un branch, una o más instrucciones después de la bifurcación se ejecutan independientemente del resultado del branch. Esos ciclos que se pierden al seguir una branch se utilizan para insertar estas instrucciones antes de la rama. Es decir, se reordena la secuencia de ejecución de las instrucciones. El compilador debe realizar un análisis de dependencia de datos para determinar si es posible la bifurcación retardada. Pueden surgir situaciones en las que no se pueda mover ninguna instrucción después de la bifurcación (a la ranura de retardo). En este caso, se coloca una NOP (no operation) después de la bifurcación. La bifurcación retrasada tiene la ventaja de un bajo costo de hardware y depende mucho del compilador para llenar las ranuras de retraso.

Otro método utilizado es la **predicción de bifurcaciones** que se basa en intentar adivinar la siguiente instrucción en el flujo de instrucciones, evitando así que el pipeline se detenga debido a la bifurcación. Si la predicción tiene éxito, no se introduce ningún retraso en el pipeline. Si la predicción no tiene éxito, el pipeline debe vaciarse y todos los cálculos causados por este error de cálculo deben descartarse. Las técnicas de predicción de bifurcación varían dependiendo de la caracterización de la bifurcación: bifurcación de control de bucle, bifurcación if/then/else, o bifurcación de subrutina.

Para aprovechar al máximo se mantiene el pipeline lleno realizándose ejecuciones especulativas: se ejecutan instrucciones antes de saber si van a ser necesarias. Si la predicción es incorrecta, se deshace el trabajo.

Existen dos tipos de predicciones: fija y variable. Las predicciones fijas son aquellas que tienen un estado más probable y se asume que eso es lo que va a ocurrir. Si la suposición es que no se toma la bifurcación, la idea es asumir que la bifurcación no ocurrirá y continuará en el camino secuencial normal. Sin embargo, el procesamiento se realiza en paralelo en caso de que se produzca la bifurcación (se prepara para una predicción incorrecta). La información de estado se guarda antes de que comience el procesamiento especulativo. Si la predicción es correcta, la información de preprocesamiento se elimina y la ejecución continúa. Si la predicción es incorrecta, el procesamiento especulativo se elimina y la información de preprocesamiento se usa para continuar en la ruta correcta.

La predicción dinámica aumenta la precisión de la predicción de bifurcaciones mediante el uso de un historial registrado de bifurcaciones anteriores. Esta información luego se combina con el código y se introduce en el predictor de rama. El componente principal utilizado para la predicción de bifurcaciones es un búfer de predicción de bifurcaciones. Los búferes de predicción de bifurcación siempre devuelven una predicción usando una pequeña cantidad de

bits. La predicción dinámica de un bit utiliza un solo bit para registrar si se tomó la última aparición de la bifurcación. La predicción de dos bits conserva el historial de las dos bifurcaciones anteriores para una instrucción de bifurcación determinada.

Varios sistemas descargan el procesamiento de predicción de bifurcaciones en circuitos especializados, que producen predicciones más oportunas y precisas.

Optimización de código

El **conteo de operaciones (operation counting)** estima la cantidad de tipos de instrucciones que se ejecutan en un ciclo y luego determina la cantidad de ciclos de máquina necesarios para cada tipo de instrucción. Esta información se puede utilizar para lograr un mejor equilibrio de instrucción. La idea es intentar de escribir los bucles con la mejor combinación de instrucciones para una arquitectura dada (ejemplo: loads, store, operaciones de enteros)

Loop unrolling (desenrollado de bucles) es el proceso de expandir un bucle de modo que cada nueva iteración contenga varias de las iteraciones originales, realizando así más cálculos por iteración de ciclo.

```
for (i = 1; i <= 30; i++)  
    a[i] = a[i] + b[i] * c;
```

When unrolled (twice) becomes:

```
for (i = 1; i <= 30; i+=3)  
{ a[i] = a[i] + b[i] * c;  
  a[i+1] = a[i+1] + b[i+1] * c;  
  a[i+2] = a[i+2] + b[i+2] * c; }
```

Si se aplica este método para secciones críticas de un programa, se podrá ver un incremento en el rendimiento, pero no sirve para cualquier sección del programa.

Otra técnica útil de optimización de bucles es la **fusión de bucles (Loop Fusion)**. La fusión de bucles combina bucles que utilizan los mismos elementos de datos. Esto puede dar como resultado un mayor rendimiento de la memoria caché, un mayor paralelismo en el nivel de instrucción y reducir la sobrecarga del bucle.

```
for (i=0; i<N; i++)  
    C[i] = A[i] + B[i];  
for (i=0; i<N; i++)  
    D[i] = E[i] + C[i];
```

results in:

```
for (i=0; i<N; i++) {  
    C[i] = A[i] + B[i];  
    D[i] = E[i] + C[i];}
```